SensiX: A System for Best-Effort Inference of Machine Learning Models in Multi-Device Environments

Chulhong Min, Akhil Mathur, Alessandro Montanari, and Fahim Kawsar, *Member, IEEE*

Abstract—Multiple sensory devices on and around us are on the rise and require us to redesign a system to make an inference of ML models accurate, robust, and efficient at the deployment time. While this multiplicity opens up an exciting opportunity to leverage sensor redundancy and high availability, it is still extremely challenging to benefit from such multiplicity and boost the runtime performance of deployed ML models without requiring model retraining and engineering. From our experience of deploying ML models in multi-device environments, we uncovered two prime caveats, device and data variabilities that affect the runtime performance of ML models. To this end, we develop an ML system that addresses these variabilities without modifying deployed models by building on prior algorithmic work. It decouples model execution from sensor data and employs two essential operations between them: a) device-to-device data translation for principled mapping of training and inference data and b) quality-aware dynamic selection for systematically choosing the execution pipeline as a function of runtime accuracy. We develop and evaluate a prototype system on wearable devices with motion and audio-based models. The experimental results show that ML models achieve a 7-13% increase in runtime accuracy solely by running on top of our system, and the increase goes up to 30% in dynamic environments. This performance gain comes at the expense of 3 mW on the host device.

Index Terms—Best-effort inference, machine learning, multi-device environments, systems

1 Introduction

Sensory connected devices are now pervasive. Mobile, wearable, and IoT devices on and around us are increasingly embracing bleeding-edge machine learning (ML) models to uncover remarkable sensory applications [1], [2], [3], [4], [5]. In this transformation, we are observing the emergence of *multi-device systems* as a natural course of multiple sensory devices surrounding us. A concrete example of this is manifested in personal devices/wearables and IoT devices deployed in our homes. Studies predict more than 9 devices per person by the year 2025 [6] with diverse sensing capabilities (e.g., motion, vision, acoustics, RF). Therefore, we believe that focusing on efficient and accurate sensing in a multi-device environment is of paramount importance moving forward.

This multiplicity is opening up an exciting opportunity to leverage sensor redundancy and high availability afforded by multiple devices, thereby enabling rich, powerful, and collaborative ML applications. For instance, once a motion model for activity recognition is deployed, a personal health tracker can selectively use a smartwatch, a smartphone, or a smart earbud to run the model. Similarly, a safety monitor in a factory can dynamically select a microphone and a camera nearby workers to detect safety-relevant events. However, the benefit from such multiplicity comes at the expense of increasing complexity. In the deployment of these applications, we identified two key caveats, *device and data variabilities* that affect the runtime performance of a deployed model and hence contribute to this complexity. They are caused by runtime factors including sensing hardware [7], resource budget, device placement [8], etc. In the example of a health tracker, each of three wearables offers identical sensing capability (e.g., motion, audio, etc.), but with contrasting runtime accuracy. A byproduct of these variabilities causes unexpected performance degradation of sensing ML models at runtime. Thus, it is imperative to take these attributes into account for designing and building a sensory AI system.

Given that these runtime variabilities can be hardly considered during the model training time, application developers often need to design a purpose-built ML model optimised for a specific device or environment, which inherently limits the coverage of the service deployment. Extensive studies have been conducted in the past to overcome these variabilities and make ML models robust at the deployment time, but they are still far from being employed in practical systems. One research thread is to retrain and re-engineer ML models by employing ML techniques such as data augmentation [9], domain adaptation, and transfer learning [10]. They enable ML models to quickly achieve satisfactory accuracy on unseen runtime factors without building a new model from scratch, but require hard assumptions such as the need for labelled data or visibility to the original model. The other thread is to leverage multi-

• *The authors are with Nokia Bell Labs, Cambridge CB3 0FA, U.K. E-mail: {chulhong.min, akhil.mathur, alessandro.montanari, fahim.kawsar}@nokia-bell-labs.com.*

Manuscript received 30 July 2021; revised 9 Mar. 2022; accepted 26 Apr. 2022. Date of publication 10 May 2022; date of current version 4 Aug. 2023. (Corresponding author: Chulhong Min.) Digital Object Identifier no. 10.1109/TMC.2022.3173914

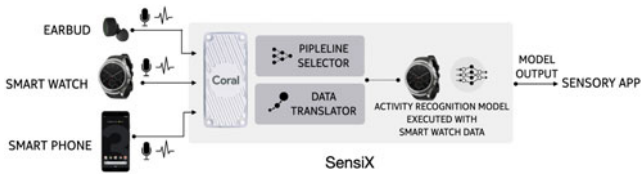


Fig. 1. SensiX stays between sensors and models and applies translation and selection operators to selectively choose the execution path for best-effort inference. Here, SensiX uses watch data and respective model to guarantee the best accuracy based on the outcome of the operators.

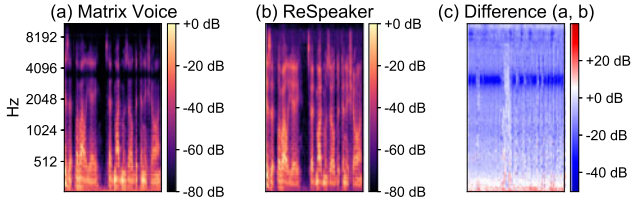


Fig. 2. Mel-spectrograms of a speech segment as captured by (a) Matrix Voice, (b) ReSpeaker, and (c) their difference.

sensor fusion that optimises the inference accuracy while addressing system issues such as time synchronisation and missing data [11], [12], [13]. However, fusion models require the tight coupling between device combinations, which makes them neither practical nor scalable in the highly fluid and dynamic environment. For example, a smartwatch can be unexpectedly turned off due to battery depletion or a new microphone can be dynamically added to the factory to expand the monitoring coverage. It would be incredibly hard to train and deploy different fusion models for all possible combinations of devices.

In contrast, in this work, we take a system-driven approach that automatically makes model inference robust and accurate at runtime. More specifically, when sensing models are given to runtime systems, we are interested in finding solutions that address device and data variabilities without model retraining or engineering, and ensuring best-effort inference under any condition in multi-device environments. Here, we define the *best-effort inference* as the optimum model accuracy guaranteed by the system given multiple execution choices.

From our experiences of deploying ML models in multi-device environments, we uncovered two hidden opportunities: a) device-to-device data translation and b) quality-aware pipeline selection. By building on prior algorithm work to realise these operations, we design and develop SensiX, a system for deploying machine learning in multi-device environments. SensiX stays between sensory devices and corresponding models, and performs principled data engineering to select the best execution path as a function of model accuracy while externalising model management and execution away from applications. SensiX achieves this with two purpose-built neural operations, as shown in Fig. 1. First, a neural translation operator deals with *device variability* by mapping data across devices. Second, a runtime quality assessment operator deals with *data variability* by selecting the right execution path for the best model accuracy. Collectively, these two operators enable SensiX to dynamically and automatically compose a model execution path under any condition to ensure best-effort inference while coping with runtime device and data variabilities.

We evaluate SensiX on two representative multi-device sensing applications built with motion and audio signals for physical activity and keyword recognition. Our results suggest that SensiX offers a 7-13% increase in overall accuracy and up to 30% increase across different environment dynamics. This performance gain comes at the expense of 3 mW on the host device, however with a significant reduction of development complexity and cost.

In what follows, we discuss the unique characteristics of multi-device systems and present systems challenges to inform our design decisions. Next, we describe the technical details of SensiX and its different operations. We then move to the evaluation of SensiX and reflect on some critical issues before concluding the paper.

2 Background and Experience

Over several decades, extensive studies on ML have been conducted to understand us and the world around us from raw sensory signals. Since these algorithms, especially deep learning-based ones, naturally require a substantial amount of computation, much effort has also been put toward enabling them to run on resource-constraint devices, e.g., using model offloading and partitioning [14], [15], model compression [16], [17]. On the basis of the insights from these works, mobile runtime systems have been studied and built, which take inference pipelines (or ML models) as the main workload and manage their performance over dynamic, heterogeneous environments, especially in terms of accuracy, energy consumption, and latency. Typically, they exploit alternative, substitutable processing options for given inference pipelines, e.g., at the level of sensor [18], [19], processor [20], approximation of model [21], [22], and dynamically select the best one based on their expected quality and the system policy.

Although the quality (and corresponding performance) of a sensing model is dynamic, the runtime assessment of those systems is mostly limited to resource metrics such as energy cost and latency. On the contrary, they relatively have a static view of the accuracy of sensing models and mostly rely on their average accuracy obtained in the training phase. This works in the conventional, single-device environment where the same device is supposed to be used. However, emerging multi-device environments bring new challenges, *device and data variabilities* that make runtime accuracy of ML models dynamic and unpredictable. We present data-driven evidence and reflect on key design challenges for ensuring best-effort inference quality in a multi-device environment.

Device Variability. Even before a sensor signal reaches the sensory application (e.g., a classifier), it passes through several processing stages including ADC conversion, DSP processing, OS processing – each of which can introduce some artefacts in the signal. Naturally, these artefacts vary across devices, as such different devices capture the same physical phenomena slightly differently. This *heterogeneity* characteristic has a profound impact on model performance, especially when different devices are available for data acquisition. To provide empirical evidence, in Fig. 2,

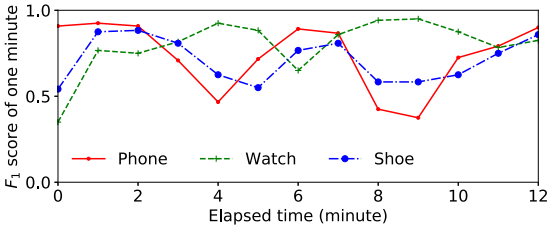


Fig. 3. Runtime accuracy of activity recognition model [11] trained with the Opportunity dataset [23]; 1) The same device offers varying accuracy over time and 2) different devices offer the best accuracy at different times.

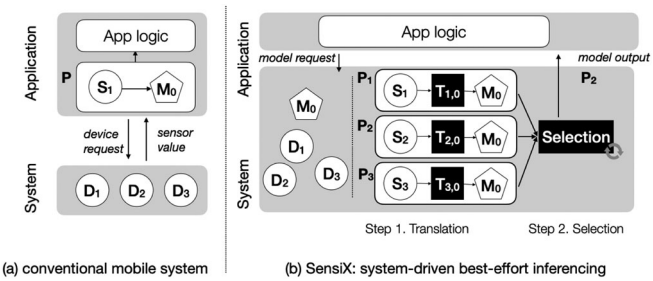


Fig. 4. System operation; D, S, M, and T represents devices, sensor data, models and translation functions, respectively.

we show mel-spectrograms of a 3-second speech segment as recorded by two microphones (Matrix Voice and ReSpeaker) simultaneously. We observe that the microphones exhibit differences in their frequency responses to the same speech input - also visualised in the rightmost figure. In [7], the impact of heterogeneous IMU sensors on the activity recognition performance has also been thoroughly studied. Since these variations are common characteristics when different, heterogeneous devices are involved in a sensing task irrespective of the modalities, unexpected performance degradation would be inevitable if a pre-trained model is deployed in unseen devices or shared with different devices. A straightforward solution would be to train device-specific models, but it would be almost infeasible considering device heterogeneity in today's market.

Data Variability: Many sensory devices around us share a standard set of sensors, e.g., IMU, microphone, etc., offering *redundant, substitutable* sensing capabilities. For example, a keyword spotting model can selectively run on one of the available microphones around a user. Similarly, an activity recognition model can be performed with one of the IMU-equipped wearables, e.g., a smartphone, a smartwatch, or even an earbud. There are several factors that constitute and affect the expected runtime accuracy, spanning hardware [7], device placement, and even users' behavioural characteristics [8], [24]. The dynamic nature of these factors makes runtime accuracy dynamic even with the fixed composition of an inference pipeline (i.e., sensor stream from the same device with the same sensing model.) To quantify this aspect, we provide empirical evidence with an activity recognition model in Fig. 3. With the Opportunity dataset [23], we selected three IMU devices placed on a hip, a left lower arm, and a right shoe, which can be mapped to the typical position of a smartphone, a smartwatch and a smart shoe, respectively. Then, we trained three models [11] separately for each device and observed how F_1 score of these models changes every minute. The results show that 1) each device offers varying runtime accuracy over time and 2) more importantly, the best performing device also changes over time.

The goodness of an ML model often determines the goodness of a sensing system. However, while the above challenges do not necessarily contribute to the goodness of an ML model at training time (which is solely dependent on the quality, quantity, and diversity of training data, training strategy, and model architecture), it is important to note that the combined effect of these factors could significantly degrade the runtime performance of ML models.

3 SensiX Design

3.1 Design Goal

We have seen in the previous section that when an ML model is deployed in a multi-device setting with heterogeneous devices from diverse manufacturers, a certain accuracy degradation, compared to the accuracy obtained in the training phase, is inevitable due to *device and data variabilities*. Recently, extensive studies have been conducted to compensate such a gap, e.g., by adopting data augmentation [9], transfer learning [10], and incremental learning. They show remarkable performance improvement without training an entirely new model, but still require a significant burden for data collection and model engineering. This is further complicated in the multi-device environment we consider given the high variability in device forms (e.g., phones, watches, wearables, speakers and other IoT devices) which are built by a plethora of manufacturers and offer diverse sensing capabilities (e.g., motion, vision, acoustics, RF). In this context, accounting for all possible combinations of devices and sensors that users might have with ad-hoc solutions based on data augmentation or model engineering is unfeasible.

In this paper, we propose a novel approach of boosting the runtime accuracy of sensing models, i.e., the *system-driven best-effort inference*. Without relying on model engineering and application modification, our approach actively intervenes between sensor data and sensing models, and achieves accuracy improvement by dynamically addressing device and data variabilities in an autonomous manner.

3.2 Separating ML Model Execution From Application

In conventional sensing systems, applications are entirely responsible for the execution of sensing models. As shown in Fig. 4a, an application requests sensor data of interest (e.g., IMU or audio data) to the system and manages the end-to-end operations required for model processing, spanning over data collection, model design and tuning, deployment, etc. While there are public model hubs available for pre-trained models such as Tensorflow Hub [25] and PyTorch Hub [26], it is still mostly developers' burden to construct the full execution pipeline from raw sensor data.

A critical aspect of *system-driven* best-effort inference is to separate the execution complexities of a sensing model from the model training process as well as from the application logic (Fig. 4b). This facet is particularly essential in several aspects. First, it can take significant, but duplicate

efforts away from application developers and model experts. As the execution of sensing models becomes a core, increasingly demanding operation in ML applications, such separation naturally becomes a key requirement of sensing systems; similar to how traditional OSes have abstracted basic tasks from computer programs such as handling I/O, controlling peripheral devices, etc. Second, it is almost infeasible for application developers and model experts to address those *runtime* properties in advance at development time and model training time. On the contrary, the system can intervene with the execution of sensing models actively and dynamically because it has more visibility and fine-grained control over device and data variabilities.

3.2.1 Level of Abstraction

Separating execution operations from application space demands articulated and useful abstraction hiding the underlying complexities. One of the design challenges is to determine at which level of sensing pipelines is abstracted. A typical pipeline of sensing models consists of sensing, preprocessing, running models, and delivering context outputs. Accordingly, we can imagine the level of abstraction corresponding to each level of operation. In this paper, we abstract the complexities of *sensing models*. That is, applications specify sensing models of interest for their logic, and SensiX takes care of the full operations required to execute the models. Our design decision in Fig. 4b offers several benefits, including application-side flexibility of choosing sensing models, a significant reduction of code complexity, and better system-wide management and coordination.

Our design choice is different from modern mobile operating systems in diverse aspects. For example, current sensor libraries, e.g., *SensorManager* [1] on Android, can be seen to provide an abstraction of *sensing*. That is, the system handles hardware operation for sensor readings, and the application takes care of the rest of the pipelines (See Fig. 4a). In this case, applications have a high level of flexibility, but it imposes a significant burden on developers at the same time. The system also has very little room for performance optimisation. In another extreme, Android also provides *high-level context*, e.g., *Activity Recognition API* [2] as another unit of abstraction. If an application specifies the context type of interest, e.g., *activity*, the system takes care of the full sensing pipeline for activity recognition and delivers the final result only. It relieves all the burden for context inference from application developers and gives the system more flexibility, e.g., choice and scheduling of sensing models, for the system-wide optimisation of the resource use. However, it limits the flexibility of application logic and requires a very well-defined and pre-established taxonomy of context vocabulary.

3.3 Dynamic Translation and Selection

To address device and data variabilities at runtime in multi-device environments, we devise two purpose-built neural operators that actively act in the middle of sensors and models, *device-to-device data translator* (Section 4.2) and *quality-aware runtime pipeline selector* (Section 4.3). First, the device-to-device data translator minimises the data variation across devices using a machine-learned component. When the model is deployed in unseen, different devices, SensiX collects unlabelled data in the background and learns the translation function with the collected data, which maps sensor data from a new device to its equivalent point in the training distribution. Second, the quality-aware runtime pipeline selector estimates the runtime quality of available pipelines (a pair of translated signals and an ML model) at runtime and selects a pipeline offering the best quality.

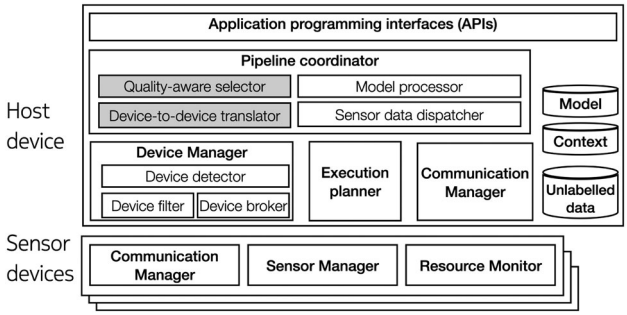


Fig. 5. System architecture of SensiX.

Fig. 4b shows the example operation of SensiX when three devices (D_1, D_2, D_3) are available and one model (M_0) trained with D_0 is given. SensiX first creates the device-to-device data translation functions ($T_{1,0}$, $T_{2,0}$, $T_{3,0}$) for all devices and generates multiple, substitutable pipelines by combining a translation function and a sensing model. For example, $T_{1,0}$ makes the data from the sensor S_1 on D_1 similar to the data from D_0 and the corresponding pipeline is generated by composing a sensor (device), a translation function, and a model, e.g., $< D_1, T_{1,0}, M_0 >$. Then, SensiX periodically assesses the runtime quality of each pipeline and then dynamically selects the best one. We put the translation operator prior to the selection operator because the translation affects the runtime quality.

4 SensiX Operation

4.1 Architecture

SensiX takes the binary of a sensing model as an input and provides applications with context outputs of the given model. Fig. 5 shows the overall architecture of SensiX. It spans over a host device and multiple sensor devices.

- *Host device:* Once sensing models are registered, the host device maintains them in its database for models. The device manager discovers suitable sensor devices for the given sensing models and keeps track of their resource availability. The *pipeline coordinator* dynamically creates an optimal execution pipeline with two primitive operations of SensiX, a neural translation operation for principled mapping of device-to-device data (Section 4.2) and a quality-aware device selection operation (Section 4.3). It also takes care of the execution of the created pipeline when the sensor data is delivered. The execution planner makes the system-wide schedule of the

1. https://developer.android.com/reference/android/hardware/SensorManager
2. https://developers.google.com/location-context/activity-recognition

execution pipelines and dynamically controls the sensing status of external sensor devices via the communication manager. Once the context information is generated, it is delivered to the application and also maintained in the database of context values if needed for the query of past data, e.g., for the retrospective summary.

- *Sensor device:* The sensor device communicates with the host device via the sensor-side communication manager. The sensor manager (de)activates the sensing task based on the execution schedule and the resource monitor monitors its resource status, e.g., remaining battery, CPU/memory usage, and transmits it to the host device periodically or when requested.

4.2 Device-to-Device Data Translation

Prior literature [27], [28] on sensing systems has established that heterogeneities in sensor data are omnipresent and can be caused by a number of issues, including variability in hardware, software or usage dynamics of the sensing devices. More critically, it has been shown that even subtle variabilities in sensor data can potentially degrade the performance of state-of-the-art ML models [29], [30]. Indeed, this poses a major challenge for multi-device sensing systems, wherein there is a very high likelihood of variability in the devices owned by a user. For instance, a user may have multiple microphone-enabled devices on (e.g., an Apple iPhone) or near their body (e.g., Amazon Echo), each of which can capture the user's speech and process it through a speech recognition model to understand the user's intent. However, due to the variations in microphone hardware and software processing pipelines across manufacturers, a model trained on an Apple iPhone microphone may not work well for an Amazon Echo.

A simple solution would be to train a separate model for each device in the system, however this would incur significant costs to collect and *label* training data for each new device that is added to the system. Instead, SensiX makes a practical and scalable choice: it assumes that there is one sensing model shared across all the devices – this model could be trained by a developer either on data collected from one of the devices or even on a separate off-the-shelf training dataset collected from one or multiple devices. Further, we assume that the developer does not provide access to the weights of this pre-trained sensing model, i.e., it is a black-box model – this enables SensiX to support both open-source and proprietary models.

Under these practical assumptions, the technical challenge is to enable highly accurate sensing on multiple heterogeneous devices, even when the sensing model may not have been trained on the same device. To this end, SensiX provides support for *device-to-device data translation* which maps (or translates) a given sensor data from any device to its equivalent point in the training distribution. This translation happens transparently at inference-time (or test-time) and aims at reducing the discrepancy between the test data and the training data on which the sensing model was trained. Overall, the device-to-device data translation aims to boost the accuracy of the pre-trained sensing model on those devices whose data distribution might differ from the training distribution. Note that when the training and test device are the same, the translation operation is not needed and is ignored by SensiX.

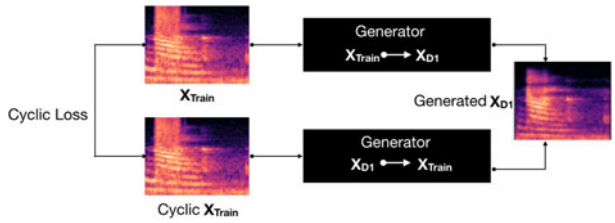


Fig. 6. Translation process for $X_{D_1} \rightarrow X_{Train}$.

The translation component is machine-learned and based on the principles of Cyclic Generative Adversarial Networks (CycleGAN) as proposed in [29], [31]. We extend this prior work to support the translation of various data modalities, including motion data from IMU (an accelerometer and a gyroscope) and audio data from a microphone.

We learn a pair-wise translation function between each user device and the training device (on which the sensing model was trained). Prior works have shown that CycleGAN models can learn these mapping between data distributions solely based on unlabelled and unpaired data, which significantly reduces the cost of training the translation model. We assume that at the time of releasing the sensing model, its developer also provides a small amount of unlabelled data X_{Train} sampled from the training distribution and it is stored on the host device. Further, when a new device D_1 is added to the multi-device ecosystem, SensiX collects a small amount of unlabelled data X_{D_1} from it with a user's permission and also sends it to the host device. This data need not be time-aligned or paired with the training data X_{Train}. Upon receiving unlabelled datasets X_{Train} and X_{D_1}, the host device initiates the training of a translation mapping $X_{D_1} \rightarrow X_{Train}$ based on the CycleGAN architecture as shown in Fig. 6. The CycleGAN architecture consists of four neural networks (2 generators and 2 discriminators) that are jointly optimised using adversarial learning – in our implementation, we use a 6-layer CNN with residual blocks to train the generators and a 4-layer CNN to train the discriminators. Note that such a training operation is conducted only once and can be offloaded to the cloud. Once the training process is done, the trained generator $X_{D_1} \rightarrow X_{Train}$ is used to perform real-time translation of the sensor data collected from device D_1 to make it similar to the training data. After the translation, it is then passed to the next operations of SensiX for further processing and computing the inferences.

4.3 Quality-Aware Pipeline Selection

After obtaining the translation function, SensiX constructs execution pipelines for each available device by combining the translation function and the ML model. Then, the next question is how to choose the right pipeline out of multiple candidates. We identify two challenges that have to be addressed to make the system practical; (1) how to quantify the runtime quality and (2) how to minimise the system cost while maximising the selection benefit?

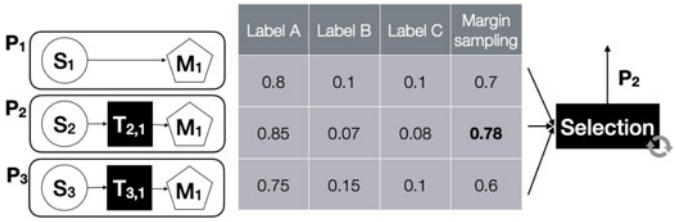


Fig. 7. Quality-aware selection; (s)ensor, (t)ranslation, (m)odel. This is an example when three devices are available and the mode trained one of these deivces is given.

Quality Quantification. There have been prior attempts to quantify the quality of sensory signals in the domain of signal processing. The most representative example is the signal-to-noise ratio (SNR) which was proposed to assess the purity of a signal. However, such signal-level quality assessment is not suitable to assess the quality of an ML-based execution pipeline, i.e., its expected runtime accuracy, because the subsequent operations (translation function and sensing model) also affect the quality of the model output. For example, the motion signals on a smartwatch in walking situations would be considered to have the high quality for movement detection, but can be seen to have the low quality for hand gesture recognition.

To address this, we present a novel pipeline-level quality assessment by adopting and modifying the heuristic-based quality assessment (HQA) method proposed in [32]. Its key idea is to leverage confidence values reported from a classifier in the sensing model for given (translated) sensory signals and quantify the quality of the execution pipeline based on these values. Confidence values represent how confident a classifier is in the inference output from a given input data. For example, a final softmax layer in neural networks produces a list of probabilities of given sensor data being a member of each class. Leveraging such characteristics, the concept of *uncertainty* of inference output was proposed in the domain of active learning, which represents how uncertainly a given inference instance is to be labelled. Inspired by this, we quantify the quality of an execution pipeline by adopting margin sampling [33]. Note that the uncertainty has no formal definition and can be estimated using different metrics such as margin sampling, highest confidence, and entropy of confidence values [34]. In active learning, the study shows that the best metric is task-dependent [35]. Thus, we used margin sampling which showed the best performance in our datasets used in the evaluations. Margin sampling is computed by taking the difference between the probabilities of the two most likely classes. We can consider that an inference instance with a higher margin is more *certain* to be labelled, compared to the other instance with a lower margin. While the uncertainty-based quality indicates the certainty of inference output from the execution pipeline, more certainty does not always guarantee more accurate inferences. However, we empirically show that, when the model is reasonably well trained, such quality-based assessment and selection contribute to achieving higher overall accuracy in Section 5.

Fig. 7 shows the operational flow and example of the quality-aware selection. SensiX gathers the sensor data from all available devices and obtains the translated data (if the device is different from the device used for training the input model). SensiX executes the model inference with the (translated) data and obtains confidence values for all pipeline candidates. Then, for each pipeline, SensiX computes the margin sampling, i.e., the difference of probabilities between its first and second most probable labels and selects the execution pipeline which shows the highest margin sampling.

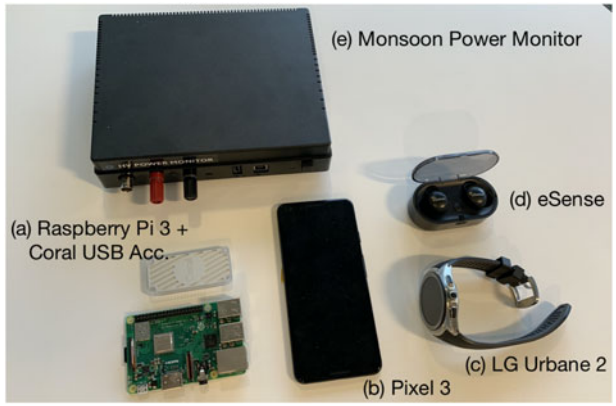


Fig. 8. Hardware setup: Personal edge as (a) Raspberry Pi 3 with Google Coral USB accelerator, sensor devices as (b) Pixel 3 smartphone, (c) LG Urbane 2 smartwatch, (d) eSense earbuds, and (e) Monsoon power monitor.

Energy-Efficient Selection. A practical issue in runtime selection is to determine a proper interval of the selection. Since the sensing quality dynamically changes even with the same topology of devices, continuous quality assessment is needed. However, our quality assessment requires all execution pipelines to be performed, i.e., sensing, translating, and model execution, thus the frequent selection could incur a significant system overhead in terms of energy and CPU. To avoid such costs and make the system practical, we adopt a widely used duty-cycling technique for the quality-aware selection. That is, by leveraging the temporal locality of human/device contexts, SensiX performs the quality assessment and selection periodically at the fixed interval and deactivates unselected pipelines until the next interval. In this paper, we empirically set the selection interval to 10 seconds, which was optimal in our dataset. Besides the periodic interval, SensiX also triggers the selection immediately if it detects system events that can affect the runtime quality, e.g., registration and deregistration of a model, and join and leave of a sensor device.

It is important to note that there are a number of system parameters (e.g., sensor types, inference models, and sensing tasks) that determine the optimal selection interval. It would also be different depending on an individual's behavioural patterns. Considering such complexity, we believe the optimal interval should be determined at runtime in an adaptive, personalised way. For example, when a new model or device is added to SensiX, it can gradually increase the interval from 1 s, estimate the overall quality considering the runtime accuracy and resource usage, and find the saturation point. We leave it as future work.

4.4 SensiX Prototype and Implementation

We implement the SensiX prototype on off-the-shelf devices. Fig. 8 shows the hardware setup. For the host

device, we used Raspberry Pi 3 with Google Coral USB accelerator and developed the host device-side components with Python and Tensorflow 1.X. For sensor devices, we considered three devices, Pixel 3 smartphone, LG Urbane 2 smartwatch, and eSense [3]) earbuds. For the Android devices, we developed the sensor-side components that run as an Android background service. eSense does not have processing capability on-board and thus we developed an eSense broker which runs on the host device.

A multi-device system naturally requires a *host* that orchestrates the runtime operations and manages the model execution. In a conventional system, a resource-rich device, e.g., a smartphone in personal sensing environments can be assigned with such functionalities. The operations discussed earlier, some of which are neural operations, demands careful system-wide orchestration and needs to be executed actively in the background. However, modern smartphones are optimised exclusively for maximising battery life, as the OSes often ignore any background operation that has relatively high energy expenses. A clear indication of such decisions of modern mobile OSes is the constrained imposed of background processing for applications. These restrictions have severe implications for the performance of a runtime orchestrator as system-wide optimisation opportunities are relatively limited. To this end, for the thorough evaluation of SensiX, we have taken a dedicated subsystem route for hosting the runtime orchestrator and implemented it on the AI accelerator. We envision that SensiX can co-exist with future smartphones (or other prominent mobile forms).

With a proliferation of multi-device sensing systems, future OSes may remove such restrictions. We consider runtime orchestration and related system operations for multi-device sensing should be designed as a specialised subsystem. Deep learning algorithms drive many sensory models today, and it is natural to expect neural accelerators will power these subsystems, which are not adopted yet in current smartphones.

This centralised orchestration and execution mean that SensiX considers every sensor device in the environment as merely a sensor stream provider equipped with a supported communication interface. However, if sensor devices afford processing capability, resource-related system cost would be expected to reduce, e.g., by using code offloading to sensors or pipeline partitioning [14], [36].

5 Evaluation

We present extensive experiments to evaluate the effectiveness of SensiX in multi-device scenarios. We use two multi-device sensing applications built with motion and audio signals for physical activity and keyword spotting, respectively. First, we investigate the effect of our device-to-device data translation and quality-aware selection mechanisms on the runtime accuracy and robustness of sensing models using the multi-device datasets. Then, we perform micro-benchmarks on top of our system prototype to understand the overheads of the SensiX operations.

5.1 Experimental Setup

Models, and Datasets. For the evaluation, we use two sensing tasks: *human activity recognition* (HAR) with IMU data and *keyword spotting* with audio data. We choose these two tasks because IMUs and microphones are core sensors in personal, multi-device environments and they are representative tasks for these sensors. For a comprehensive analysis, we used the multi-device datasets and conducted repetitive experiments with different combinations of system parameters and comparison groups.

HAR: For human activity recognition, we develop a deep learning model proposed in [11], which employs a CNN-based feature extractor with 4 residual blocks containing 2 convolutional layers each; it takes 1-second-long data as an input. The model has two fully-connected layers and an output layer. For the analysis, we use the *RealWorld* dataset [37]. It consists of sensor data recorded from 15 participants with seven smartphones on their body. Each participant performs eight physical activities; walking, running, sitting, standing, lying, stairs up, stairs down, and jumping. For the experiments, we select 3-axis accelerometer and 3-gyroscope data from three devices deployed on a forearm, a head, and a thigh, each of which represents the typical position of a smartwatch, an earbud, and a smartphone, respectively; the sampling rate is 50 Hz.

Keyword Spotting. We use the keyword detection architecture proposed in [38]. It takes a two-dimensional tensor extracted from the 1-second-long audio recording (time frames on one axis and MFCC on the other axis) as an input. The architecture consists of two convolutional layers, a global average pooling layer, and a fully-connected layer. For training and testing, we use the *Keyword* dataset [29] which consists of 65,000 speech keywords re-recorded at 16 kHz on three different embedded microphones (Matrix Voice, ReSpeaker, USB microphone) simultaneously; each file has a 1-second long spoken keyword which belongs to one of 31 keyword classes.

We split all datasets to ensure independence between the training and test set. For the additional details, refer to [11], [38] for the architecture of sensing models and [29], [37] for the datasets.

Sensor Workloads. We consider two types of sensor workloads, *static* and *dynamic*. In the static workload, all devices are available all the time. It is to investigate the overall performance of SensiX and the baselines. The dynamic workload is to study the robustness of the system in dynamic situations where some devices become temporarily unavailable, e.g., the battery runs out or a watch is left on a desk. We generate the four dynamic workloads with the following *availability probability* values, 0.7, 0.8, 0.9, 1.0, respectively. For each workload, the availability of each device (either available or unavailable) is randomly decided based on the probability value. The decision is made independently to other devices.

Comparison. SensiX offers high-accuratey, robust sensing in the runtime environment with two main operations, device-to-device data translation and quality-aware pipeline selection. To identify the impact of each operation, we considered the following baselines which also include variations of SensiX itself:

- Single-avg: The traditional practice in context monitoring is to use a single, fixed device for a sensing model, e.g., either a smartphone, a smartwatch, or an

earbud at runtime as shown in Fig. 4a. Thus, for each sensing task, we can consider three cases using different devices. Single-avg reports the average performance of these three cases.

- SensiX-native: SensiX-native is built on top of the device discovery, but does not have the capability of the device-to-device data translation and quality-aware selection. Note that, by bringing AI execution to the system layer, SensiX performs device discovery in the background, e.g., using Bluetooth and Wi-Fi scanning, and dynamically maps sensing models to available devices. SensiX-native selects the device for model processing in a round-robin manner out of available devices.
- SensiX-trans: SensiX-trans adopts the translation operation on top of SensiX-native, but does not have the quality-aware selection. That is, the device for model processing is selected in a round-robin manner, but the translation operation is added and used if the given model was not trained in the selected device.
- SensiX-QS: SensiX-QS selects the device with the quality-aware selection, but without the device-to-device data translation.

Model and Translation Configuration. We demonstrate the capabilities of SensiX in several training configurations where pre-trained models are available only from a subset of the devices. For the HAR task, we assume that two model instances (trained with data from *head* and *forearm* devices, respectively) are given and SensiX trains one translation function ($thigh \rightarrow head$). Indeed, as there is no model trained for *thigh*-worn sensors, SensiX performs translation on the data collected from the thigh to make it resemble the data from a training device (e.g., *head*), before passing this data to the *head* model for inference. Similarly, for the keyword spotting model, we assume that one model instance (trained with the data from Matrix Voice) is given and SensiX trains translation functions for the other two devices (ReSpeaker → Matrix Voice and USB microphone → Matrix Voice). During training of each sensing model, we use a held-out validation dataset to evaluate model accuracy and chose the model with the lowest validation loss. While training the translation model, we used a quantitative metric commonly employed in generative modelling literature called Peak Signal-to-noise ratio (PSNR). PSNR is closely related to the mean square error and is suggestive of the distance between the datasets before and after translation. The model which provided the highest PSNR during the training process was selected as the final translation model. To keep the evaluation focused, we only show results on models pre-trained on data from a single device (e.g., *head*). However, note that SensiX data translation does not make any assumption that the source model is pre-trained only on data from one device. Prior works have shown the capabilities of data translation in multi-device settings [29].

Performance Metrics. As a key performance metric, we consider the runtime accuracy of sensing models in the multi-device setting and compare SensiX against the aforementioned baselines. Since our datasets are imbalanced, we use the micro-averaged F_1 score [39]. In the dynamic workloads, the execution of a model cannot be supported if the dedicated device in Single-avg is unavailable or all the devices are unavailable in the SensiX variations. To reflect such scenarios, we set the F_1 score to 0 during those moments when ML execution is not supported.

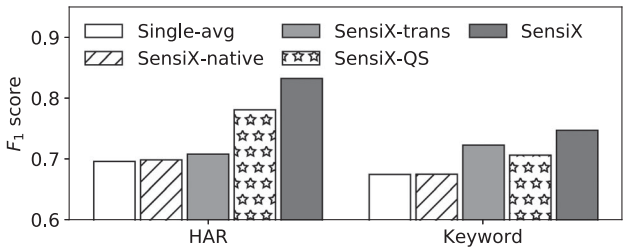


Fig. 9. Overall performance.

5.2 Effect of SensiX on Accuracy Improvement

5.2.1 Overall Performance

Fig. 9 shows the overall performance under the static workload. The results show that SensiX increases the overall average F_1 score of sensing models by up to 0.13 without modifying them. More specifically, for the HAR task, SensiX achieves 0.83 of F_1 score, whereas the average F_1 score of the cases when a single device is used without any SensiX operations (Single-avg) is 0.70. It shows that the device-to-device data translation and quality-aware selection mechanisms enable ML models to have more accurate results. We break down the performance by looking into and comparing the results of SensiX variations. As expected, Sensis-native shows similar performance to Single-avg because SensiX-native selects each device in turn, thus its performance converges to the average performance of the cases when each device is used. In HAR task, the improvement from the device-to-device translation, i.e., SensiX-trans (0.71) compared to SensiX-native (0.70), is not meaningful. This is because the performance of the translated *thigh* data with the *head* model is still much lower than the other two device cases. However, SensiX-QS shows 0.78 of F_1 score, 8% higher than SensiX-native, which shows the effectiveness of the quality-aware selection when multiple IMU devices are available. Also, when the translation operation is used together with the quality-aware selection, SensiX further achieves a 5% higher F_1 score than SensiX-QS. This shows that, even though the overall performance of the translation operation (from thigh data to head data) is not meaningful, our selection mechanism well spots the moments when the translated thigh data outperforms the data from the other two devices, and contributes to achieving the higher overall performance.

As shown in Fig. 9, the keyword spotting model shows a similar trend. SensiX increases the overall average F_1 score by 7%; the F_1 score of SensiX and Single-avg is 0.75 and 0.68, respectively. Note again that, in the keyword spotting task, only one model trained with the data from Matrix Voice was used and thus ReSpeaker and USB microphone suffer poor accuracy when the translation operation is not applied, due to the heterogeneity issue described in § 4.2. Different from the HAR case, we can observe the significant contribution of the translation operation. By adopting the

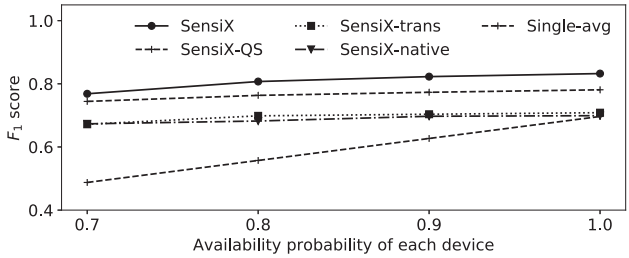


Fig. 10. Performance of HAR in dynamic workloads.

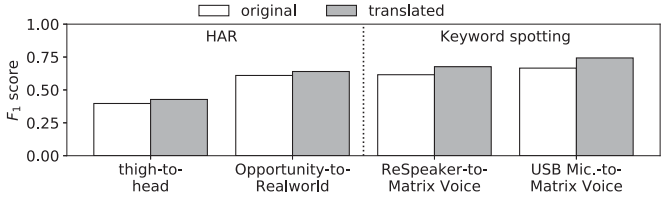


Fig. 12. Effect of translation operation: (A) HAR (left) and (b) keyword spotting (right).

translation function only, SensiX-trans achieves 0.72 of F_1 score.

It is important to note that the main goal of our work is not to train the most accurate sensing models and achieve the high accuracy of each model, but to show that SensiX operations can increase the accuracy of pre-trained models in a multi-device sensing system. Our results, i.e., the relative improvement Single-avg to SensiX, clearly demonstrate this capability of SensiX.

5.2.2 Robustness in Dynamic Workloads

Fig. 10 shows the F_1 score of the HAR model while increasing the availability probability from 0.7 to 1.0. When the probability is 0.7, each device is available with the probability of 0.7; for example, in this case, the probability of having two available devices out of three is 0.441 ($_3C_2 \times 0.7 \times 0.7 \times 0.3$). The results show that the SensiX capability with device discovery in multi-device environments achieves a higher level of robustness of model execution. As expected, Single-avg shows poorer performance as the availability probability becomes lower; it shows a linear relationship to the availability probability. However, the decrease of F_1 scores of SensiX variations is not significant. SensiX variations fail to deliver the model output only when all devices are unavailable, which is very unlikely when there are multiple devices. For example, the F_1 score of SensiX is 0.77, 0.81, 0.83, and 0.83 when the availability probability is 0.7, 0.8, 0.9, and 1.0, respectively.

Fig. 11 shows the result of the keyword spotting model in the dynamic workload. The results show a similar trend to the HAR model, but the decrease of F_1 score when the availability probability goes down from 1.0 to 0.7 is much smaller, e.g., F_1 score of SensiX is 0.75 and 0.72 with 1.0 and 0.7 of the probability, respectively. This is because the accuracy of keyword models is not much different across the devices when the translation operation is applied.

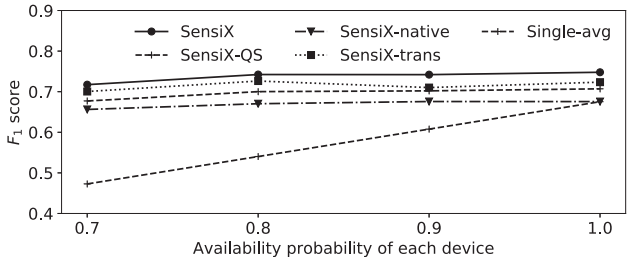


Fig. 11. Performance of keyword in dynamic workloads.

5.3 In-Depth Analysis

5.3.1 Device-to-Device Translation

We look deeper into how much the translation operation improves the model performance.

Translation Between IMU Sensors. Here, we report two cases of translation between IMU sensors: (1) placement-to-placement translation, i.e., between the same type of IMU sensor, but with different placements and (2) device-to-device translation, i.e., between the different type of IMU sensor but with same placements. For the former, we report the performance of the *thigh*-to-*head* translation which was used in the experiments. For the latter, we introduce the Opportunity dataset [23] and investigate the performance of the translation from the *left lower arm* device in Opportunity to the *smartwatch on a forearm* in RealWorld, which can be seen to be placed in the same position. Since the collection configuration is different between Opportunity and RealWorld, we resample and normalise the Opportunity data before the translation; the sampling rate and accelerometer range are 30 Hz and $\pm 3g$ for Opportunity and 50 HZ and $\pm 2g$ for RealWorld. Fig. 12a shows the accuracy improvement when the translation operation is applied. More specifically, for the *thigh* case, the performance of the *original* and *translated* is reported when the alternative model (here, *head* model) is used with the original *thigh* data and the translated *thigh*-to-*head* data, respectively. The results show that our translation operation improves the F_1 score from 0.39 and 0.43. In the case of the translation of watch-worn devices from Opportunity to RealWorld, F_1 score of the original and translated is 0.61 and 0.64, when the model trained with the *forearm* device in RealWorld is used with the resampled and normalised data from *left lower arm* in Opportunity, without and with the translation, respectively.

One may argue that, even with the accuracy improvement, translation between IMU sensors can be seen as impractical due to their poor absolute accuracy. However, our experimental results in Fig. 9 show that, together with the quality-aware selection, the translation provides the meaningful improvement of the system-wide accuracy. Considering that having device/placement-specific models requires a significant burden for data collection and model engineering, SensiX can be used to complement applications when a new device is added, until the dedicated model is available. We believe we can further optimise the translation performance by adopting recent studies on domain adaptation for IMU sensors and motion models, e.g., [40], [41].

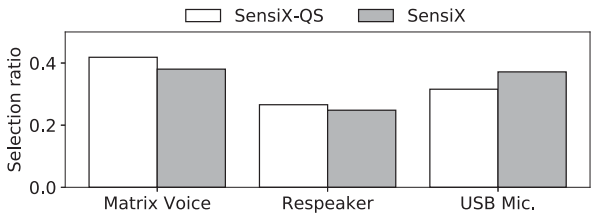


Fig. 13. Ratio of selection (keyword spotting).

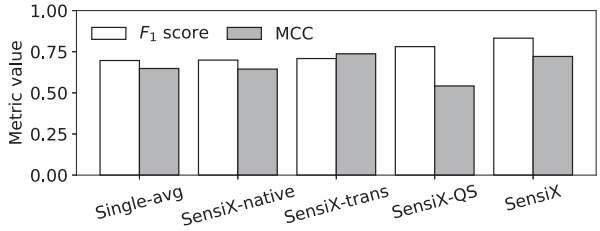


Fig. 14. Comparison between F_1 score and MCC (HAR).

Translation Between Microphones. Fig. 12b shows the results for the keyword spotting model. We assume that a model pre-trained on *Matrix Voice* microphone is provided and is now tested with data from *ReSpeaker* and *USB Mic.*. As such, SensiX performs the following translation operations: *ReSpeaker*-to-*Matrix Voice* and *USB Mic.*-to-*Matrix Voice*. We observe that due to translation, the F_1 score of ReSpeaker increase from 0.62 to 0.68 (i.e., 6% increase) and for USB Mic., the F_1 score increases from 0.67 to 0.74 (i.e., 7% increase). These accuracy gains are significant for the Keyword Spotting model trained for Matrix Voice, whose best F_1 score is 0.77 when it is trained and tested on the same microphone. In other words, the translation operation is able to recover 40% (for ReSpeaker) and 70% (for USB Mic.) of the drop in F_1 score of the Keyword Spotting model due to microphone variability.

5.3.2 Quality-Aware Selection

To have a deeper understanding of the behaviour of the quality-aware selection, we look into how often each device is selected by the selection operator in the static workload. Fig. 13 shows the selection ratio of three microphones on the keyword spotting model. The difference of ratio patterns between SensiX-QS and SensiX represents how the selection decision changes when the translation functions are added; note again that SensiX-QS does not have the translation operation. The results show that our selection mechanism well reflects the runtime quality, in two different ways. First, when the original signal is used without the translation (i.e., SensiX-QS), MatriX Voice is selected the most frequently, mainly due to its relatively higher performance. Second, on SensiX, the performance of both ReSpeaker and USB Mic. improves from the translation to Matrix Voice as shown in Fig. 12b. However, interestingly, while USB Mic. is selected more often, ReSpeaker is less selected. This is mainly because translated USB Mic. achieves comparable performance to Matrix Voice and thus produces more chances of being selected, especially instead of ReSpeaker.

5.3.3 Further Analysis on an Imbalanced Dataset

Here, we present a further analysis on an imbalanced dataset using the Matthews correlation coefficient (MCC), introduced by biochemist B. W. Matthews in 1975 [42] for biomedical research. The MCC metric is also widely used in the field of machine learning because it is generally regarded as a balanced measure that can be used for imbalanced datasets [43]. The MCC can be calculated from the confusion matrix using the following formula:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$$

, where TP, TN, FP, and FN are the number of true positives, true negatives, false positives, and false negatives, respectively. The MCC value ranges from -1 to 1; -1 and 1 represent an inverse and perfect prediction, respectively, and 0 presents an average random prediction.

Fig. 14 shows the F_1 score and MCC of the HAR task under the static workload. The experimental results show that, while two metrics show different patterns depending on the baselines, we observe that SensiX also increases MCC by up to 0.07 without modifying sensing models. MCC of SensiX is 0.72, whereas MCC of Single-avg (i.e., when a single device is used without any SensiX operations) is 0.65. Interestingly, we observe that SensiX-QS shows a relatively lower MCC than other baselines. It is mainly because the decision from the quality-aware pipeline selection decreases the number of true negatives. However, MCC increases again when the device-to-device data translation is adopted (i.e., SensiX). It again shows that the two main operations of SensiX complement each other and contribute to achieving higher overall performance. We omit the results of the keyword spotting task because the trend of F_1 score and MCC was similar.

5.4 Micro Benchmark of SensiX Prototype

To understand the system behaviours, we conduct the micro-benchmark of the SensiX prototype with off-the-shelf devices. Fig. 15 shows the main operations of SensiX. We first study the resource characteristics of model processing and then examine the system overhead of SensiX. We measure the energy cost using a Monsoon power monitor.

Model Execution. As described in Section 4.4, sensor devices act as a data source and model execution is conducted on a host device. Thus, the main status of the sensor device is either *deactivated* (idle mode), or *activated* (sensing and streaming data). Table 1 shows the power profiles of sensor devices for the HAR task, which are obtained from the Monsoon power monitor; we omit the result for the keyword spotting task due to the page limit. We report the power cost of sensing and BLE

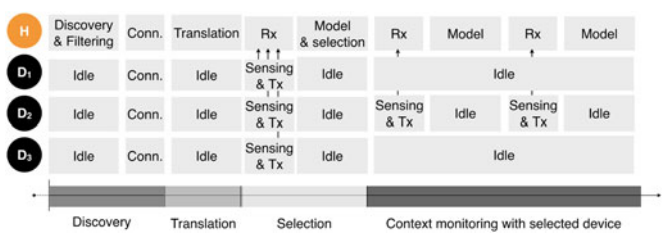


Fig. 15. Main operations of SensiX; (H)ost device and sensor (d)evices. An example when D_2 is selected.

TABLE 1
Power Cost for the Motion Model on Sensor Devices

Operation	Power (mW)		
	Pixel	LG Urbane 2	eSense
Idle	28.1	27.8	6.6
Sensing	7.2	8.6	5.0 (BLE)
Bluetooth Tx	177.1	68.5	

TABLE 2
Resource Cost for Model Processing on a Host Device

Model	Parameters	Energy	Inference time
HAR	385 k	0.86 mJ	1.93 ms
Keyword	1,846 k	28.12 mJ	62.17 ms

Tx for eSense together because its firmware does not support those functions separately. The transmission cost for Pixel 3 and LG Urbane is relatively much higher than that of eSense because Bluetooth classic is used for communication with these Android devices. We expect that further energy saving can be achieved if the communication is developed on top of BLE.

The network usage is different depending on the sensing task. On each device, the HAR and keyword spotting tasks send 300 and 32 k bytes per second, respectively.

The main operations of the host device are to receive sensor data and execute the model processing. Table 2 shows the energy cost and inference time to be taken to process an instance of AI model execution on Raspberry Pi 3 and Coral USB accelerator. We observe that the execution of the keyword spotting model takes more energy and a longer time due to its bigger size of the architecture. The average power to receive motion and audio data via Bluetooth classic remains around 10 mW. The memory usage of the HAR and keyword spotting models is 1.84 MB and 3.9 MB.

System Overhead. The major operations of SensiX beyond model execution are as follows. (1) SensiX discovers nearby devices by periodically establishing the Bluetooth connection with paired devices. The time to be taken to establish the connection is measured as 0.9 sec and 5.7 sec for Android devices and eSense, respectively. We believe that the shorter time for Android devices comes from their optimisation of Bluetooth stack. (2) Once a new device is added and the corresponding model is not available, SensiX generates the translation function either locally or remotely on the cloud. Once the translation model is available, SensiX processes it when sensor data from the added device is received. The average time for the translation model for one sample, i.e., one-second-long sensor data is 20 ms and 480 ms for the HAR and keyword model, respectively. Their execution takes much longer than the model execution because the cycle GAN network in the translation model is much more complex. (3) SensiX performs the quality-aware selection at the interval of the duty cycle. For one selection operation, the overhead is to receive the sensor data and perform the sensing model for all devices during the assessment window (1 sec.), i.e., additional 1.7 mJ and 28.30 mJ for the HAR and keyword spotting task, respectively; when three devices are available, additional processing of two models is needed. Considering the selection interval, 10 secs, the additional power overhead for the model execution is 0.17 mW and 2.83 mW. The additional energy cost for the network transmission is negligible because sensor data is sent altogether at once when a one-second-long segment is gathered, not whenever a sensor sample is generated and the transmission is also made once every 10 seconds. After the model execution, SensiX selects the execution pipeline, which is expected to have the best runtime quality. Due to the simplicity of margin sampling computation, the execution time and energy cost are negligible, < 1 ms and < 1 mJ, respectively.

6 Discussion

Why not Multi-Device Fusion?. We assume that sensing models are built using sensor data from a *single* sensory device. Recently, a number of studies on multi-sensory fusion have been conducted to maximise the inference accuracy while addressing potential system issues such as time synchronisation and missing data [11], [12], [13], [44], [45]. While these works contributed substantially to achieving higher performance, we believe that they are not practical, yet to be used at the personal-edge in the multi-device environments. First, the fusion model requires all the devices involved in the training, to be activated all the time at runtime, thereby incurring significant system costs. Second, more importantly, considering the dynamics of multi-device environments, different fusion models are needed to be built and trained for all possible combinations of devices, which may not be feasible. For example, the fusion model trained with a smartphone and a smartwatch may be useless if a user forgets to wear the watch. Similarly, a new model will be needed if a user buys a new wearable device or an IoT device around a user becomes available.

Beyond the Accuracy. SensiX can be easily extended to consider resource-related runtime metrics by adopting online profiling tools for energy [46] and transmission latency [47] or by leveraging the benchmark study of the model performance, e.g., [48], [49]. To this end, SensiX allows the policy to be specified as a cost function and selects the pipeline with the minimum cost output. For example, for the policy of minimising the total energy consumption, the corresponding cost function can be defined as $f(D, M) = total\_energy\_cost(D, M)$, where D and M are a device and a sensing model, and total_energy_cost() is a function that returns the expected total energy consumption of devices when D is selected for the processing of M. Several factors can be considered together by defining a cost function as their weighted sum.

Distributed System Architecture. In this paper, we implemented the system orchestrator part of SensiX on a dedicated host device, and we envision this could be a smartphone in the future. This has the advantage that all personal data remains local to the user, with significant advantages in terms of privacy protection [50]. An alternative solution would be to implement this functionality on the cloud and make it accessible from SensiX clients. In addition to stricter privacy and security measures to avoid leaking personal data, this approach would require careful consideration of the latency and energy cost that might be introduced. In fact, the cloud service might not always be available due to the overload of the remote servers or due to connectivity issues. Moreover, streaming sensor data to the cloud could severely affect

battery lifetime, especially for high-rate sensors. We leave the study of these tradeoffs for future work.

Generalisability of Proposed Techniques. In this paper, we mainly focus on personal-edge environments with IMU and microphone sensors. A typical context for this is one that considers multiple mobile devices and wearables (e.g., phone, watch, earbuds, ring) and diverse IoT devices (e.g., smart speakers, TVs) owned by the same person or the same household. In this case, the need for device translation when new devices join the group or device selection due to the dynamic conditions of daily living is higher. However, since device and data variabilities are common characteristics also in other multi-device environments (e.g., industrial settings), we believe that the main features of SensiX, (a) separating execution of sensing models from application space and (b) proposing device-to-device translation and quality-aware selection, are still valid in other edge environments with a different type of sensors. One such example is when multiple cameras with overlapping fields of view are deployed to provide collaborative video analytics services. Due to deployment constraints, cameras might be produced by different manufacturers while the sensing models assume a specific make and model. In this situation, SensiX's device-to-device translation is of paramount importance to ensure the effective use of the cameras. Similarly, when covering a very large area, the cameras will not all be subject to the same conditions (e.g., fog, smoke, dirt on lens), hence the quality-aware selection offered by SensiX could guarantee optimal performance even in diverse and challenging situations. However, translation and quality-assessment algorithms might need to be implemented differently depending on the environment and sensor type due to their different characteristics. Also, even in multi-device environments with IMU, microphone, and cameras, SensiX would not work well if they do not capture the same context, e.g., wearables owned by different users, far-placed microphones, and cameras without overlapping fields of view. Thus it will be important to quantify the feasibility of SensiX when it is deployed in new environments. We leave it as future work.

Alternatives to Inference-Time Data Translation. Our approach of CycleGAN-based data translation at inference time was motivated by two key design goals: a) SensiX should neither require re-training of the sensing model nor assume that the model developer provides access to the weights of the pre-trained sensing model, b) SensiX should not assume the availability of labelled data at inference time. We note that in the absence of these assumptions, there are alternative approaches to transfer knowledge from a source domain to a target domain, e.g., [51] proposed translating the source domain data to the target domain and retrain the sensing model. Further, transfer learning approaches such as model fine-tuning can be used to adapt the original sensing model to the target device, if adequate labelled data is available from the target device. Both these approaches violate the assumptions and design goals of SensiX, hence we did not employ them.

Model Monitoring and Update. An important issue related to the deployment of ML models on edge devices is the need to perform continuous monitoring and necessary updates to the model. Among other factors, the need for updating the models can be necessitated by the change in underlying data distribution of the sensor devices, e.g., due to a faulty sensor or significant variations in how the device is worn or used. In such cases, both the sensing and translation models would need to be updated. Although exploring this topic was out of scope of this paper, we highlight two broad questions that can be studied in future work. First, we need a way to monitor the data distribution of each device and determine that a significant distribution shift has happened. To this end, both data-driven approaches or techniques based on monitoring system events could be explored. Second, once a distribution shift is detected, how do we trigger the retraining of the translation and sensing models? Should the retraining operation run as a background process and once the models are retrained, they replace the existing models. Or should the system inform the user that the models are no longer valid and cause application downtime while the models are being updated? We believe all of these are important future research topics for SensiX and edge ML in general.

7 Related Work

Context-Aware Middleware for Body Sensor Networks (BSNs). There have been research efforts to develop context-aware middleware platforms for BSNs. They have developed abstractions to tackle challenges associated with context retrieval, device discovery, and user mobility, thereby making it easier to develop context-aware applications. One main, relevant direction is the dynamic sensor selection work [18], [52], [53]. Grounded on the understanding of the effect of different characteristics on recognition accuracy, e.g., sensor type and composition, and device placement, they dynamically select the best sensor with the objective of optimising a system policy, e.g., maximising accuracy, minimising energy cost. While they have presented execution strategies for various purposes, their consideration of the runtime accuracy has been limited. They mostly assumed that the runtime accuracy is static while the associated devices are available and made the decision based on the average accuracy.

Our work contributes to this rich body of BSN research by offering novel systematic aspects and addressing the dynamic nature of runtime accuracy. First, SensiX automatically generates sensing pipelines to make a given sensing model work in new, unseen devices by presenting device-to-device translation [29]. Second, by adopting the selection mechanism [32], SensiX ensures best-effort inference of ML models.

Support for ML Model Inference. A number of model serving platforms have been proposed to facilitate the model deployment into production and enhance the runtime performance, but they mostly focus on the inference latency. TensorFlow Serving is a serving system developed by Google for TensorFlow models [54]. Crankshaw *et al.* presented Clipper, an online prediction serving system that deploys ML models in separate containers and reduces prediction latency by employing caching and batching technique [55]. InferLine has been proposed to meet the tight end-to-end latency requirements by optimally configuring ML prediction pipelines [56]. Unlike these work, SensiX primarily targets the runtime accuracy as the main performance metric of ML models and proposes novel, practical system-driven solutions that address device and data variabilities without model modification.

Streaming Architectures. From the point of view that sensor devices act as a data source, streaming processing architectures like Amazon Kinesis [57] and Apache Kafka [58] can be seen similar to SensiX. Those systems are built as cloud-side messaging queue systems to deal with a high volume and number of streaming data durably, reliably, and with scalability. In this regard, they mainly focus on the data management layer, lacking the system aspects such as heterogeneity management for sensing quality. SensiX targets the execution pipeline layer to achieve accurate and robust sensing on top of the data management layer.

8 Conclusion

We presented SensiX, a purpose-built runtime component for the personal edge to offer best-effort inference in a multi-device sensing environment. SensiX sits between the sensor devices and the corresponding sensory models in a personal edge device and comprises of two neural operators. A device-to-device data translation operator and a quality-aware device selection operator to cope with the device and data variabilities while externalising the model management and execution away from the applications. The combination of these two operators enables SensiX to boost the runtime accuracy of sensory models in a multi-device sensory environment. We discussed different design cardinals, operational principles and implementation details of SensiX. We demonstrated the efficacy of SensiX through extensive testing with real-world motion and acoustic sensing workloads, including three different public datasets and two different models. Our evaluation highlighted the ability of SensiX in boosting the overall runtime accuracy of different sensing models in multi-device sensing applications by 7-13% and up to 30% increase across different environment dynamics. We showed that this gain comes at the minimal expense of 3 mW on the personal edge device, however with a significant reduction of development complexity and cost.

In the current version of SensiX, we assume that application developers provide model binaries. However, we envision that public repositories for pre-trained sensing models such as [25], [26] can be easily used with SensiX, thereby allowing developers to choose and execute pre-trained models efficiently. We anticipate such ability of SensiX will enable application developers to focus on the application logic and model developers to focus on accurate model design.

In our future avenue of work, we plan to deploy and evaluate SensiX in a real-world personal edge environment with multiple sensory devices. Besides, we want to explore the applicability of SensiX with other modalities, and in particular for vision-based applications. Finally, we aspire to assess the efficacy of SensiX in reducing code complexity by engaging developers in building multi-device sensing systems.

References

[1] E. Miluzzo *et al.*, "Sensing meets mobile social networks: The design, implementation and evaluation of the cenceme application," in *Proc. 6th ACM Conf. Embedded Netw. Sensor Syst.*, 2008, pp. 337–350.

[2] Y. Li, A. Miller, A. Liu, K. Coburn, and L. J. Salazar, "Acoustic measures for real-time voice coaching," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2020, pp. 2755–2763.

[3] F. Kawsar, C. Min, A. Mathur, and A. Montanari, "Earables for personal-scale behavior analytics," *IEEE Pervasive Comput.*, vol. 17, no. 3, pp. 83–89, Jul.–Sep. 2018.

[4] Y. Shen, M. Voisin, A. Aliamiri, A. Avati, A. Hannun, and A. Ng, "Ambulatory atrial fibrillation monitoring using wearable photoplethysmography with deep learning," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2019, pp. 1909–1916.

[5] A. Ferlini, A. Montanari, C. Min, H. Li, U. Sassi, and F. Kawsar, "In-ear PPG for vital signs," *IEEE Pervasive Comput.*, vol. 21, no. 1, pp. 65–74, Jan.–Mar. 2021.

[6] B. Safaei, A. M. H. Monazzah, M. B. Bafroei, and A. Ejlali, "Reliability side-effects in Internet of Things application layer protocols," in *Proc. 2nd Int. Conf. System Rel. Saf.*, 2017, pp. 207–212.

[7] A. Stisen *et al.*, "Smart devices are different: Assessing and mitigatingmobile sensing heterogeneities for activity recognition," in *Proc. 13th ACM Conf. Embedded Netw. Sensor Syst.*, 2015, pp. 127–140.

[8] C. Min, A. Mathur, A. Montanari, and F. Kawsar, "An early characterisation of wearing variability on motion signals for wearables," in *Proc. 23rd Int. Symp. Wearable Comput.*, 2019, pp. 166–168.

[9] T. T. Um *et al.*, "Data augmentation of wearable sensor data for parkinson's disease monitoring using convolutional neural networks," in *Proc. 19th ACM Int. Conf. Multimodal Interact.*, 2017, pp. 216–220.

[10] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, Oct. 2010.

[11] L. Peng, L. Chen, Z. Ye, and Y. Zhang, "AROMA: A deep multi-task learning based simple and complex human activity recognition method using wearable sensors," *Proc. ACM Interactive Mobile Wearable Ubiquitous Technol.*, vol. 2, no. 2, pp. 74:1–74:16, Jul. 2018.

[12] S. Yao, S. Hu, Y. Zhao, A. Zhang, and T. Abdelzaher, "DeepSense: A unified deep learning framework for time-series mobile sensing data processing," in *Proc. 26th Int. Conf. World Wide Web*, 2017, pp. 351–360.

[13] Y. Vaizman, N. Weibel, and G. Lanckriet, "Context recognition in-the-wild: Unified model for multi-modal sensors and multi-label classification," *Proc. ACM Interactive Mobile Wearable Ubiquitous Technol.*, vol. 1, no. 4, pp. 168:1–168:22, Jan. 2018.

[14] E. Cuervo *et al.*, "MAUI: Making smartphones last longer with code offload," in *Proc. 8th Int. Conf. Mobile Syst. Appl. Serv.*, 2010, pp. 49–62.

[15] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan, "Towards wearable cognitive assistance," in *Proc. 12th Annu. Int. Conf. Mobile Syst. Appl. Serv.*, 2014, pp. 68–81.

[16] N. D. Lane *et al.*, "DeepX: A software accelerator for low-power deep learning inference on mobile devices," in *Proc. 15th Int. Conf. Inf. Process. Sensor Netw.*, 2016, pp. 1–6.

[17] S. Liu, Y. Lin, Z. Zhou, K. Nan, H. Liu, and J. Du, "On-demand deep model compression for mobile devices: A usage-driven model selection framework," in *Proc. 16th Annu. Int. Conf. Mobile Syst. Appl. Serv.*, 2018, pp. 389–400.

[18] S. Kang *et al.*, "Orchestrator: An active resource orchestration framework for mobile context monitoring in sensor-rich mobile environments," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun.*, 2010, pp. 135–144.

[19] A. Bakar, T. Rahman, A. Montanari, J. Lei, R. Shafik, and F. Kawsar, "Logic-based intelligence for batteryless sensors," in *Proc. 23rd Annu. Int. Workshop Mobile Comput. Syst. Appl.*, 2022, pp. 22–28.

[20] P. Georgiev, N. D. Lane, K. K. Rachuri, and C. Mascolo, "LEO: Scheduling sensor inference algorithms across heterogeneous mobile processors and network resources," in *Proc. 22nd Annu. Int. Conf. Mobile Comput. Netw.*, 2016, pp. 320–333.

[21] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, "MCDNN: An approximation-based execution framework for deep stream processing under resource constraints," in *Proc. 14th Annu. Int. Conf. Mobile Syst. Appl. Serv.*, 2016, pp. 123–136.

[22] A. Montanari, M. Sharma, D. Jenkus, M. Alloulah, L. Qendro, and F. Kawsar, "ePerceptive: Energy reactive embedded intelligence for batteryless sensors," in *Proc. 18th Conf. Embedded Netw. Sensor Syst.*, 2020, pp. 382–394.

[23] D. Roggen *et al.*, "Collecting complex activity datasets in highly rich networked sensor environments," in *Proc. 7th Int. Conf. Netw. Sens. Syst.*, 2010, pp. 233–240.

[24] M. Kreil, B. Sick, and P. Lukowicz, "Dealing with human variability in motion based, wearable activity recognition," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops*, 2014, pp. 36–40.

[25] Tensorflow models & dataset, Accessed: Feb. 9, 2020. [Online]. Available: https://www.tensorflow.org/resources/models-datasets

[26] Pytorch hub. Accessed: Jul. 1, 2020. [Online]. Available: https://pytorch.org/hub/

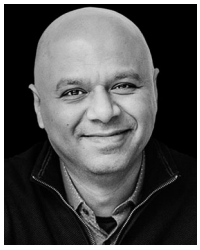
[27] A. Das, N. Borisov, and M. Caesar, "Do you hear what I hear?: Fingerprinting smart devices through embedded acoustic components," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2014, pp. 441–452.

[28] S. Dey, N. Roy, W. Xu, R. R. Choudhury, and S. Nelakuditi3, "Accelprint: Imperfections of accelerometers make smartphones trackable," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2014, pp. 23–26.
[29] A. Mathur, A. Isopoussu, F. Kawsar, N. Berthouze, and N. D. Lane, "Mic2Mic: Using cycle-consistent generative adversarial networks to overcome microphone variability in speech systems," in *Proc. 18th Int. Conf. Inf. Process. Sensor Netw.*, 2019, pp. 169–180.
[30] A. Mathur *et al.*, "Using deep data augmentation training to address software and hardware heterogeneities in wearable and smartphone sensing devices," in *Proc. 17th ACM/IEEE Int. Conf. Inf. Process. Sensor Netw.*, 2018, pp. 200–211.
[31] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 2223–2232.
[32] C. Min, A. Montanari, A. Mathur, and F. Kawsar, "A closer look at quality-aware runtime assessment of sensing models in multi-device environments," in *Proc. 17th Conf. Embedded Netw. Sensor Syst.*, 2019, pp. 271–284.
[33] T. Scheffer, C. Decomain, and S. Wrobel, "Active hidden Markov models for information extraction," in *Advances in Intelligent Data Analysis*, F. Hoffmann, D. J. Hand, N. Adams, D. Fisher, and G. Guimaraes ed., Berlin, Germany: Springer, 2001, pp. 309–318.
[34] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, no. 3, pp. 379–423, 1948.
[35] C. Körner and S. Wrobel, "Multi-class ensemble-based active learning," in *Proc. Eur. Conf. Mach. Learn.*, 2006, pp. 687–694.
[36] R. Newton, S. Toledo, L. Girod, H. Balakrishnan, and S. Madden, "Wishbone: Profile-based partitioning for sensornet applications," in *Proc. 6th USENIX Symp. Netw. Syst. Des. Implementation*, 2009, pp. 395–408.
[37] T. Sztyler and H. Stuckenschmidt, "On-body localization of wearable devices: An investigation of position-aware activity recognition," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun.*, 2016, pp. 1–9.
[38] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," 2018, *arXiv:abs/1804.03209*.
[39] V. Van Asch, "Macro-and micro-averaged evaluation measures," *Belgium: CLiPS*, vol. 49, 2013.
[40] Y. Chang, A. Mathur, A. Isopoussu, J. Song, and F. Kawsar, "A systematic study of unsupervised domain adaptation for robust human-activity recognition," *Proc. ACM Interactive Mobile Wearable Ubiquitous Technol.*, vol. 4, no. 1, pp. 1–30, Mar. 2020.
[41] C. Chen *et al.*, "Motiontransformer: Transferring neural inertial tracking between domains," in *Proc. AAAI Conf. Artif. Intell.*, 2019, pp. 8009–8016.
[42] B. W. Matthews, "Comparison of the predicted and observed secondary structure of t4 phage lysozyme," *Biochimica et Biophysica Acta Protein Struct.*, vol. 405, no. 2, pp. 442–451, 1975.
[43] S. Boughorbel, F. Jarray, and M. El-Anbari, "Optimal classifier for imbalanced data using matthews correlation coefficient metric," *PLoS One*, vol. 12, no. 6, 2017, Art. no. e0177678.
[44] F. J. Ordóñez and D. Roggen, "Deep convolutional and LSTM recurrent neural networks for multimodal wearable activity recognition," *Sensors*, vol. 16, no. 1, 2016, Art. no. 115.
[45] S. Yao, Y. Zhao, S. Hu, and T. Abdelzaher, "Qualitydeepsense: Quality-aware deep learning framework for Internet of Things applications with sensor-temporal attention," in *Proc. 2nd Int. Workshop Embedded Mobile Deep Learn.*, 2018, pp. 42–47.
[46] A. Pathak, Y. C. Hu, and M. Zhang, "Where is the energy spent inside my app?: Fine grained energy accounting on smartphones with Eprof," in *Proc. 7th ACM Eur. Conf. Comput. Syst.*, 2012, pp. 29–42.
[47] A. Gönther and C. Hoene, "Measuring round trip times to determine the distance between WLAN nodes," in *Proc. Int. Conf. Res. Netw.*, 2005, pp. 768–779.
[48] M. Antonini, T. H. Vu, C. Min, A. Montanari, A. Mathur, and F. Kawsar, "Resource characterisation of personal-scale sensing models on edge accelerators," in *Proc. 1st Int. Workshop Challenges Artif. Intell. Mach. Learn. Internet Things*, 2019, pp. 49–55.
[49] J. McChesney, N. Wang, A. Tanwer, E. de Lara, and B. Varghese, "DeFog: Fog computing benchmarks," in *Proc. 4th ACM/IEEE Symp. Edge Comput.*, 2019, pp. 47–58.
[50] A. Montanari, A. Mashhadi, A. Mathur, and F. Kawsar, "Understanding the privacy design space for personal connected objects," in *Proc. 30th Int. BCS Hum. Comput. Interact. Conf.*, 2016, pp. 1–13.
[51] W. Luo, Z. Yan, Q. Song, and R. Tan, "PhyAug: Physics-directed data augmentation for deep sensing model transfer in cyber-physical systems," in *Proc. 20th Int. Conf. Inf. Process. Sensor Netw.*, 2021, pp. 31–46.
[52] P. Zappi *et al.*, "Activity recognition from on-body sensors: Accuracy-power trade-off by dynamic sensor selection," *Wireless Sensor Networks*, R. Verdone ed., Berlin, Germany: Springer, 2008, pp. 17–33.
[53] M. Keally, G. Zhou, G. Xing, J. Wu, and A. Pyles, "PBN: Towards practical activity recognition using smartphone-based body sensor networks," in *Proc. 9th ACM Conf. Embedded Netw. Sensor Syst.*, 2011, pp. 246–259.
[54] Tensorflow serving. Accessed: Feb. 9, 2021. [Online]. Available: https://www.tensorflow.org/tfx/guide/serving
[55] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: A low-latency online prediction serving system," in *Proc. 14th USENIX Symp. Netw. Syst. Des. Implementation*, 2017, pp. 613–627.
[56] D. Crankshaw *et al.*, "InferLine: latency-aware provisioning and scaling for prediction serving pipelines," in *Proc. 11th ACM Symp. Cloud Comput.*, 2020, pp. 477–491.
[57] Amazon kinesis. Accessed: Jul. 1, 2020. [Online]. Available: https://aws.amazon.com/kinesis
[58] Apache kafka. Accessed: Jul. 1, 2020. [Online]. Available: https://kafka.apache.org/

Chulhong Min received the PhD degree in computer science from KAIST, in 2016. He is currently a research scientist with Nokia Bell Labs, Cambridge, U.K. In 2017, he joined Nokia Bell Labs. His research interests include mobile and wearable systems, on-device AI, and edge computing. He is an associated editor for ACM Proceedings on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT), and also on the program committees of various primer conferences.

Akhil Mathur received the master's degree in computer science from the University of Toronto, and the PhD degree in computer Science from University College London. He is currently a principal research scientist with Nokia Bell Labs, Cambridge, UK. He is on the editorial board of ACM Proceedings on Interactive, Mobile, Wearable and Ubiquitous Technologies, and was a committee member for leading mobile and sensor systems venues. His research has been covered by several media organizations, including the New Yorker, Financial Times, Livemint, and Canadian Broadcasting Corporation.

Alessandro Montanari received the master's degree (*cum laude*) in computer engineering from the University of Bologna, Italy, and the PhD degree in computer science from the University of Cambridge, U.K. He is currently a senior research scientist with Nokia Bell Labs, Cambridge, U.K. His current research interests include embedded systems, earables for vital signs sensing, and ultra-low power applied machine learning. He is committee member of leading mobile and sensor systems conferences and journals.



Fahim Kawsar (Member, IEEE) currently leads Pervasive Systems Research with Nokia Bell Labs, Cambridge. He holds a mobile systems professorship in computing science from the University of Glasgow. He studies the forms and intelligence of emerging mobile, IoT, wearable devices.